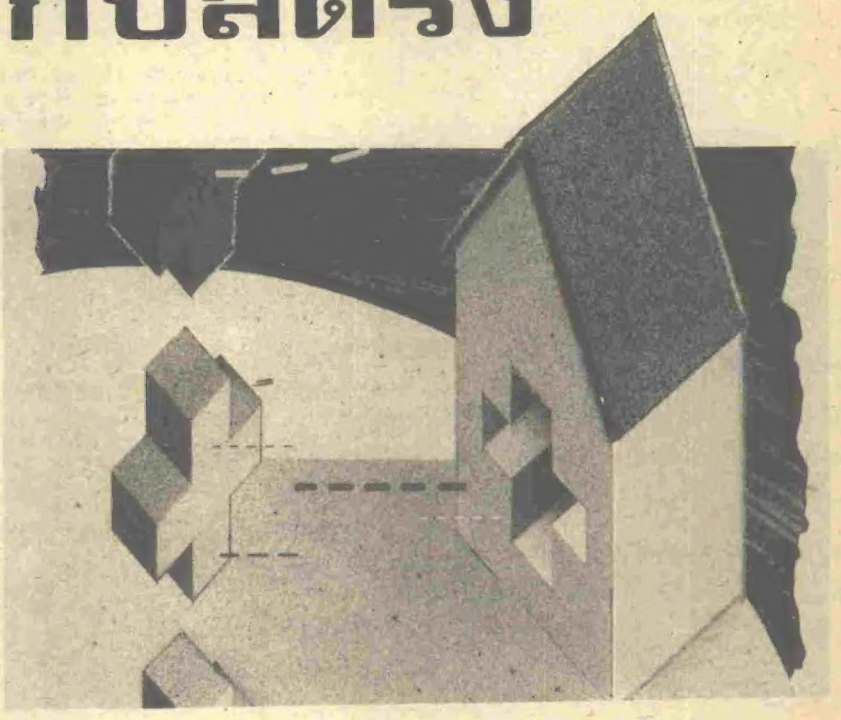
SOFT MAIL

ซอฟต์แวร์ชิพ : ดำเนินการกับสตริง



แสดงการเชื่อมต่อและสับเปลี่ยนโปรแกรมประหนึ่งว่าโปรแกรมยูนิตหนึ่ง ๆ เป็นไอซีตัวหนึ่ง

จากสกุล Collection ตามที่กล่าวในฉบับก่อน ผมจะนำมาสร้างเป็นสกุล StrCollect เพื่อใช้ในการดำเนินการกับสตริงโดยเฉพาะ แต่ความเป็นจริงแล้วเป็นการเตรียมการไปสู่โปรแกรมเอดิเตอร์โดยตรง ไม่มีความจำเป็นใดที่จะต้องกำหนดสกุลใหม่เพื่อใช้งานเฉพาะกับสตริงเท่านั้น เพราะสามารถใช้งานสกุล Collection ได้อยู่แล้ว

สำหรับในส่วนของสกุล Collection เองก็สามารถนำไปใช้งานได้มากพอสมควร เช่นใช้ในการจัดเรียงลำดับข้อมูล ก็เพียงแต่เขียนโปรแกรมสำหรับการเรียงลำดับข้อมูลเท่านั้น โดยไม่ต้องพะวงถึงการจัดเก็บข้อมูล (แต่ต้องรู้จักคำที่ใช้ใน Collection เช่น AddItem เป็นต้น) หากโอกาสอำนวยเมื่อจบจากซอฟต์แวร์ชิพแล้วผมจะนำมาเสนอต่อไป

ดังนั้นวัตถุประสงค์ที่แท้จริงของการกำหนดสกุล StrCollect ก็เพื่อใช้กับสตริงในงานอีดิต (ขอให้ดูการใช้งานเอดิเตอร์ของเทอร์โบปาสคาลเป็นหลัก) ซึ่งงานอีดิตเป็นงานที่กว้างพอสมควรและวัตถุประสงค์ของผมในการเขียนบทความนี้ก็เพื่อแสดงการเชื่อมต่อและสับเปลี่ยนโปรแกรมประหนึ่งว่าโปรแกรมยูนิตหนึ่ง ๆ เป็นไอซีตัวหนึ่งๆ ผมจึงแยกงานออกเป็น 2 สายและกำหนดชื่อเรียกดังนี้คือ

วินโดว์อีดิต จะดำเนินการทั้งปวงอันเดียวกับคีย์บอร์ดและจอภาพ จนได้ผลเป็นสตริงส่งไปให้โปรแกรมอีกส่วนหนึ่ง มีสกุลที่กำหนดไว้แล้วคือ WindowNorm กับ WindowMove และที่จะกำหนดต่อไปคือ LineEdit กับ WindowEdit จึงใช้คำว่าวินโดว์อีดิต

เท็กซ์อีดิต จะดำเนินการกับเท็กซ์คือข้อความที่ผู้ใช้ป้อน เฉพาะที่อยู่ในหน่วยความจำ คือตามที่วินโดว์อีดิตจะส่งมาเป็นสตริงหรือกลุ่มของสตริง มีสกุลที่กำหนดไว้แล้วคือ Collect และที่จะกำหนดต่อไปคือ StringCollect, TextEdit กับ PreEdit แต่ขอใช้คำว่าเท็กซ์อีดิตเป็นคำรวม

หน้าที่ของ StrCollect

ในแต่ละสกุลดังกล่าวมานี้ผมได้แบ่งมอบหน้าที่ให้ ซึ่งจะขอนำไปกล่าวในตอนต่อไป สำหรับในส่วนของ StrCollect ได้มอบให้ทำหน้าที่เกี่ยวกับการเก็บข้อมูลคือเท็กซ์กับการดำเนินการเป็นบล็อกและเป็นบรรทัด กล่าวคือ

- การลอกข้อความ (copy) จากตำบลที่หนึ่งไปยังอีกตำบลที่หนึ่ง (ของเดิมยังอยู่)
- การย้ายข้อความ (move) จากตำบลที่หนึ่งไปยังอีกตำบลที่หนึ่ง (ของเดิมถูกลบ)
- การลบข้อความ (delete) เป็นการถอนข้อความทั้งบล็อกออกจากเท็กซ์
- การดำเนินการเป็นบรรทัด คือการถอนข้อความทั้งบรรทัด (เช่นเมื่อผู้ใช้กดคีย์ Ctrl-Y) และการแทรกบรรทัดว่าง (คีย์ Ctrl-N)

ซึ่งงานเก็บข้อมูลเป็นเรื่องของ Collection อยู่แล้ว (เว้นที่จะเขียนเสริม) ดังนั้นงานหลักของ StringCollect จึงเหลือแต่การดำเนินการเป็นบล็อกและเป็นบรรทัด

กำหนดตัวแปร

เมื่อกล่าวถึงการเขียนโปรแกรมในแบบโอโอพีในสกุล Collection ผมได้กล่าวไว้ว่าลำดับแรกของการเขียนโปรแกรมในแบบโอโอพีคือการกำหนดกิจกรรม ไม่เป็นจริงเสมอไปครับเช่นในกรณีนี้งานที่ได้รับมอบบ่งกิจกรรมค่อนข้างชัดเจน สิ่งที่ต้องดำเนินการเป็นลำดับแรกคือการกำหนดตัวแปรเพื่อรองรับออปเจ็ค

ออปเจ็คของสกุลนี้คือบล็อก (block) ของข้อความตามแต่ผู้ใช้จะกำหนด (ด้วยการกดคีย์ Ctrl-K, B และ Ctrl-K, K) ดังนั้นจึงต้องมีตัวแปรสำหรับเก็บขอบเขตของบล็อก (ดังรูปที่ 1) ได้แก่ sttlne กับ sttchn เพื่อหมายต้นบล็อก และ aftlne กับ aftchn เพื่อหมายท้ายบล็อกเช่น หากมีข้อความว่า

abcdefghijklmn

หากผู้ใช้กดคีย์หมายต้นบล็อกที่ d และหมายท้ายบล็อกที่ k ข้อความในบล็อกคือ d ถึง j (ไม่รวม k เช่นเดียวกับในเทอร์โบปาสคาล)

โดยที่ sttchn และ aftchn หมายอักขระต้นบล็อกและท้ายบล็อกดังกล่าวมา ส่วน sttlne หมายถึงบรรทัดที่เป็นต้นบล็อก สำหรับ aftlne หมายถึงบรรทัดท้ายบล็อก โดยมีเงื่อนไขว่า ถ้าบล็อกอยู่ในบรรทัดเดียวกัน sttlne และ aftlne จะมีค่าเดียวกัน ถ้าบล็อกมีหลายบรรทัดจะขึ้นกับตำแหน่งของ aftchn คือ ถ้า aftchn อยู่กลางบรรทัด aftlne จะเป็นบรรทัดที่เป็นท้ายบล็อก แต่ถ้า aftchn อยู่ต้นบรรทัด (มีค่าเป็น 1) บรรทัดก่อน aftlne จะเป็นบรรทัดท้ายบล็อก (ตามแบบของเทอร์โบปาสคาลครับ)

สำหรับ dstlne กับ dstchn หมายถึงตำบลที่ปลายทางที่จะนำต้นบล็อกไปลง เมื่อมีการลอกหรือย้ายบล็อกไปลง และจะเป็นต้นบล็อกใหม่ด้วย

```
UNIT  STRCLCT;

Interface
uses  crt,prime,editroot,collect;

TYPE

    StrCollect = OBJECT(Collection)
       dstlne,sttlne,aftlne : Word;
       dstchn,sttchn,aftchn : Byte;
```

รูปที่ 1

มองภาพการจัดเก็บ

ผมมองภาพการจัดเก็บข้อมูลคือข้อความที่ผู้ใช้ป้อนนี้เหมือนกับการบันทึกบนกระดาษต่อเนื่อง มีจำนวนบรรทัดตามแต่ผู้เขียนโปรแกรมคนต่อไปจะกำหนด (ขยายได้โดยอัตโนมัติ) ในแต่ละบรรทัดคือแถวของอักขระ

(ในการกำหนดตัวแปร ส่วนมากจะให้ประกอบด้วยอักษร lne เมื่อหมายถึงบรรทัดและ chn เมื่อหมายถึงอักขระและใช้แทนคำว่าบรรทัดและอักขระด้วย)

ตัวแปรที่ใช้ในการเก็บข้อมูลคือ buff อยู่ในสกุล Collection เป็นพอยน์เตอร์ไปยังอะเรย์ของเรคอร์ด ที่ประกอบด้วยฟิลด์ dptr เป็นพอยน์เตอร์ไปยังข้อมูลที่ต้องการเก็บในกรณีนี้คือสตริง กับฟิลด์ size เพื่อเก็บขนาดของข้อมูล ซึ่งในกรณีของสตริงไม่จำเป็นแต่ผมก็นำมาใช้และต้องใช้ด้วย

สำหรับสตริงสามารถเก็บอักขระได้สูงสุด 255 ตัว แต่จะให้เก็บได้เพียง 254 ตัว เพื่อใช้ตัวเลข 255 ในความหมายของจนสุดสตริง เช่นให้ตัดอักขระจากตำแหน่งของเคอร์เซอร์ไปจนสุดสตริง เพื่อไม่ต้องหาความยาวของสตริงมาอ้างอิง

ส่วนการเก็บสตริงแต่ละตัวผมจะให้เก็บเท่ากับขนาด (ความยาวบวกหนึ่ง) ของสตริงนั้น ๆ เท่านั้นเพื่อประหยัดหน่วยความจำ และสำหรับบรรทัดที่เป็นบรรทัดว่าง (null สตริง) จะไม่จัดหน่วยความจำในฮีพให้ จะให้ชี้ไปที่ตัวแปร nullLn ซึ่งเป็นโกลบอล และให้ค่าเป็นสตริงว่าง เพื่อประหยัดอีกเช่นกันครับ ที่ทำได้เช่นนี้เพราะได้ออกแบบการจัดเก็บข้อมูลไว้เป็นพอยน์เตอร์ไปยังข้อมูลแบบใดก็ได้จึงไม่ขึ้นกับขนาดของตัวแปรสตริง

และผมขอเรียนให้ทราบว่าในโปรแกรมนี้จะมีแต่พอยน์เตอร์ไปยังสตริงเท่านั้น มีตัวแปรสตริงแท้ ๆ น้อยมาก ดังนั้นคำว่าสตริงขอให้หมายถึงสตริงในฮีพ เช่นกล่าวว่ากำหนดสตริงตัวใหม่ให้หมายถึงการกำหนดหน่วยความจำในฮีพสำหรับสตริงตัวใหม่เป็นหลัก

การเขียนคำอธิบาย

บทความตอนนี้เช่นเดียวกับบท

ความในชุดซอฟต์แวร์ชิพตอนอื่น ๆ คือ ผมจะขอนำโปรแกรมมาลงเพียงบางส่วนเท่านั้น เพื่อให้จบบทความชุดนี้ได้เร็ว และผมเห็นว่า

- โปรแกรมในเรื่องนี้ไม่ซับซ้อน ผู้ที่รู้ภาษาปาสคาลแล้วสามารถเขียนโปรแกรมได้ด้วยตนเอง

- ผู้ที่กำลังศึกษาภาษาปาสคาลอาจใช้บทความของผมเป็นโจทย์ทดลองเขียนโปรแกรม และดูเฉลยได้โดยใช้บริการซอฟต์เมล์ในฉบับนี้

และต้องการให้ผู้ที่ประสงค์จะใช้โปรแกรมในชุดนี้ (จากซอฟต์เมล์) สามารถนำไปใช้ได้โดยตรง โดยไม่ต้องศึกษาซอร์สโปรแกรม

ดังนั้นในแต่ละกิจกรรมผมจะต้องแยกหน้าที่และการทำงานของมัน ออกมาให้เห็นเด่นชัด แต่เท่าที่ผ่านมาปนอยู่กับคำอธิบายครับ ซึ่งผมจะแยกเขียนคำอธิบายต่างหากก็ไม่ได้ จึงคิดว่าวิธีนี้น่าจะดีที่สุดคือ ภายใต้หัวข้อที่เป็นกิจกรรมหากย่อหน้าใดเป็นคำอธิบายผมจะใช้เครื่องหมาย * นำหน้า แต่ก็มีบางตอนเหมือนกันที่จำเป็นต้องปะปนกันผมจะเขียนคำอธิบายไว้ในวงเล็บ ขอได้โปรดทราบตามนี้ด้วยครับ

พิจารณากิจกรรมเดิม

ก่อนที่จะกำหนดกิจกรรมใหม่จะต้องสำรวจดูกิจกรรมเดิมก่อนว่า มีกิจกรรมใดบ้างที่จะนำมาใช้และกิจกรรมใดบ้างที่เมื่อนำมาใช้แล้วจะทำให้เกิดปัญหาหรือต้องปรับปรุง (ซึ่งจะต้องกำหนดกิจกรรมทับ) พบว่าจะใช้กิจกรรมเดิมในสกุล Collection เกือบทั้งหมด และมีกิจกรรม InsertItem กับ DeleteItem ที่จะมีปัญหากล่าวคือ

ในการทำงานเป็นบล็อกได้เก็บ

```
CONSTRUCTOR  Init(numbItem,memreq : Word);
function     InsertItem(pstn,si : Word; sptr : StrPtrT) : Boolean;
function     DeleteItem(pstn : Word) : Boolean;
function     GetSize(pstn : Word) : Word;
procedure    SetSizeZero(lne : Word);

function     AddLine(sptr : StrPtrT) : Word;
function     InsertLine(pstn : Word; sptr : StrPtrT) : Boolean;
function     ReplaceLine(pstn : Word; sptr : StrPtrT) : Boolean;

procedure    SetCurPstn(lne : Word; chn : Byte);
procedure    SetStartBlock(lne : Word; chn : Byte);
procedure    SetAftBlock(lne : Word; chn : Byte);
```

รูปที่ 2

ตำแหน่งต้นและท้ายบล็อกไว้ในตัวแปรดังกล่าวมา เมื่อมีการแทรก (insert) หรือถอน (delete) ข้อมูลทั้งบรรทัดหน้าบล็อก จะทำให้บล็อกเลื่อนถอยหลังหรือไปข้างหน้าตามลำดับ แต่ค่าในตัวแปรไม่เปลี่ยนทำให้คลาดเคลื่อน

ดังนั้นจะต้องเขียนโปรแกรมสำหรับกิจกรรมทั้งสองนี้ใหม่ หรือกล่าวอีกนัยหนึ่งคือจะต้องกำหนดกิจกรรมเพื่อการนี้ใหม่และจะต้องใช้ชื่อเดิม ผมขอเรียนย้ำครับว่าจะต้องใช้ชื่อเดิม เพื่อป้องกันไม่ให้ใช้กิจกรรมเดิมกับออปเจ็คใหม่นี้คือบล็อกได้ (โดยไม่ตั้งใจ)

นอกจากนี้ยังมีกิจกรรมเดิมบางงานที่ผมเห็นว่าควรกำหนดขึ้นใหม่ เพื่อให้เหมาะสมกับการใช้งานกับสตริง จึงขอนำมารวมกล่าวในหัวข้อนี้ตามรูปที่ 2 คือ

Init

ทำหน้าที่เตรียมตัวแปรทั้งในสกุลเดิม และของตนเองโดย

- เรียกใช้ Init ใน Collection เพื่อเตรียมตัวแปรในสกุลเดิมให้เรียบร้อยก่อน โดยผ่านค่า numbItem (จำนวนบรรทัด) ไปให้

- ให้ sttlne มีค่าเป็น 1 และ aftlne มีค่าเป็นศูนย์ (เพราะได้กำหนดเงื่อนไขไว้ว่า ถ้า sttlne > aftlne แสดงว่าไม่มีการหมายบล็อก)

- เรียกใช้ CheckMem โดยส่งค่า memreq ไปให้ เพื่อตรวจสอบว่ามีหน่วยความจำเหลือพอในการทำงานต่อไปหรือไม่ (หากไม่พอ CheckMem จะให้ค่า errnum เป็น 501 ซึ่งจะให้เป็นหน้าที่ของ PreEdit เป็นผู้ดำเนินการดังจะกล่าวต่อไป)

*ที่ผมให้ตรวจสอบหน่วยความจำเมื่อเริ่มต้น และต่อไปในทุกวงรอบการทำงาน เพื่อให้แน่ใจว่าจะมีหน่วยความจำเพียงพอในวงรอบการทำงานหนึ่ง ๆ จะได้ไม่ต้องตรวจสอบทุกครั้งที่มีการเพิ่ม แทรก หรือลอกข้อความที่ต้องกำหนดตัวแปรสตริงในฮีพใหม่ อันจะเป็นเหตุให้ต้องเขียนโปรแกรมยาวและเสียเวลาในการรัน

*เพื่อการนี้จึงได้กำหนดตัวแปร ok เป็นแบบบูลีนและเป็นโกลบอล เพื่อใช้กับกิจกรรมเดิมของ Collection ที่เป็นฟังก์ชันให้ค่าเป็นบูลีน ทั้งนี้เพราะไม่ได้นำค่าซึ่งจะต้องเป็นจริงเสมอไปดำเนินการใด

*ค่าใน memreq คือจำนวนหน่วยความจำที่โปรแกรมทางด้านวินโดว์อีดิต

ต้องการใช้ เพราะได้กำหนดไว้ว่าจะให้ Init ด้านเท็กซ์อีดิตก่อน สำหรับค่าที่ StrCollect ต้องการจะกล่าวในกิจกรรม CheckMem

DeleteItem

InsertItem

กิจกรรมทั้งสองนี้จะดำเนินการถอนหรือแทรกข้อมูล (ตามลำดับ) กับ buff ทั้งบรรทัด โดย

- เรียกใช้กิจกรรมชื่อเดียวกันใน Collection เพื่อดำเนินการถอน/แทรกข้อมูลก่อน
- ตรวจว่าตำแหน่งที่ถอน/แทรกอยู่หน้าบล็อก (มีค่า lne น้อยกว่าหรือไม่) ถ้าอยู่ข้างหน้าจะลดหนึ่ง/เพิ่มหนึ่งค่าใน sttlne และ aftlne ทั้งคู่ตามลำดับ

GetSize

เป็นกิจกรรมที่เป็นฟังก์ชันให้ค่าขนาด (ความยาวบวกหนึ่ง) ของสตริง ณ บรรทัดที่ pstn โดยอ่านค่า size มาให้แก่ GetSize โดยตรง

AddLine

InsertLine

ReplaceLine

ดำเนินการลอกค่าจากสตริงที่ส่งมาไปเก็บ (ต่อท้าย) แทรก หรือแทนที่สตริงตามลำดับ โดย

- เรียกใช้ CopyStr เพื่อกำหนดสตริงขึ้นใหม่ในฮีพ แล้วลอกค่าจากสตริงที่ผ่านไปให้แก่สตริงใหม่นี้
- เรียกใช้กิจกรรม AddItem, InsertItem หรือ ReplaceItem ตามกรณี ซึ่งสำหรับ AddItem และ ReplaceItem จะเป็นกิจกรรมในสกุล Collection

```
1234567890123456789012345678901234567890123456789012345678901234567890
2aaaaaaaaaaa<aA>AAAAAAAAAAAAAAAAAAAAAAAA2
3oooooooooooooooooooooooooooooooooooooooooo3
4oooooooooooooooooooooooooooooooooooooooooooooooooooo4
5bbbbbbbbbbbbbbbb<bB>BBBBBBBBBBBBBBBBBBB5
6cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc6
7cccccccccccccccccccccccccccccccccccc7
8xxxxxxxxxxxxxxxxxxxxx<xX>XXXXXXXXXXXXXXXXXXXXX8
9............()............................9
10.......(n-nnnnnnnnnn-n).........o....10
11yyyyyyyyyyyyyyyyyyyyyyyyyy<yY>YYYYYYYYYYYYYYYYYYYYYYYYYY11
1234567890123456789012345678901234567890123456789012345678901234567890

14  THIS IS THE LAST LINE OF TEXT
```

รูปที่ 3 แสดงข้อความ (สมมติ) ที่ผู้ใช้ป้อนและได้หมายบล็อกไว้คือ

มี sttlne อยู่ที่บรรทัดที่ 8 และมี sttchn อยู่ที่ X (ใหญ่) ตัวแรก

มี aftlne อยู่ที่บรรทัดที่ 11 และมี aftchn อยู่ที่ Y (ใหญ่) ตัวแรก

และจะให้ตำบลที่ปลายทางคือ dstlne อยู่ที่บรรทัดที่ 2 และ dstchn อยู่ที่อักษร A (ใหญ่) ตัวแรก

มีคำที่ใช้ในบทความดังนี้

- ตอ คือข้อความตั้งแต่ 2 จนถึง a (เล็ก) ตัวสุดท้าย
- เศษของตอ คือข้อความตั้งแต่ A (ใหญ่) ตัวแรกไปจนสุดบรรทัด
- ท่อนหัว คือข้อความตั้งแต่ X (ใหญ่) ตัวแรกไปจนสุดบรรทัด
- เศษของหัว คือข้อความตั้งแต่ 8 จนถึง x (เล็ก) ตัวสุดท้าย
- ท่อนหาง คือข้อความตั้งแต่ 11 จนถึง y (เล็ก) ตัวสุดท้าย
- เศษของหาง คือข้อความตั้งแต่ Y (ใหญ่) ตัวแรกไปจนสุดบรรทัด

*ทั้งนี้เพราะในการดำเนินงานอาจต้องอ่านข้อมูลจากตัวแปรตัวอื่นเช่น ในการอ่านไฟล์อาจกำหนดตัวแปรสตริงเพียงตัวเดียว อ่านข้อมูลครั้งละ 1 สตริง นำไปเก็บแล้วอ่านใหม่ด้วยตัวแปรเดิม

SetCurPstn

SetStartBlock

SetAftBlock

เพื่อการให้ค่าแก่ dstlne กับ dstchn, sttlne กับ sttchn และ aftlne กับ aftchn ตามลำดับ เพื่อให้เป็นไปตามหลักการเขียนโปรแกรมในแบบโอโอพีที่ให้เข้าถึงข้อมูลผ่านกิจกรรม แต่ความจริงแล้วผมไม่ได้เคร่งครัดในเรื่องนี้เท่าใดที่จัดกิจกรรมเหล่านี้ไว้ก็เพื่ออำนวยความสะดวกเท่านั้นครับ

การดำเนินการเป็นบล็อก

เพื่อกำหนดกิจกรรมหลักในการดำเนินการกับบล็อก ผมเห็นว่าหากได้ศึกษาการดำเนินการเป็นบล็อกจากซอฟต์แวร์ที่มีอยู่เดิมจะทำให้ทำงานได้เร็วขึ้น และที่ใกล้มือคือเอดิเตอร์ของเทอร์โบปาสคาล

ตามรูปที่ 3 สมมติว่าเป็นข้อความที่ผู้ใช้ป้อนและได้หมายบล็อกไว้ กับได้กำหนดตำบลที่ปลายทางไว้โดยให้ dstlne อยู่ตรงบรรทัดที่ 2 และ dstchn อยู่ตรงอักษร A (ใหญ่) ตัวแรก จะดำเนินการกับบล็อกได้ดังนี้คือ (ขอให้อ่านคำอธิบายใต้รูปประกอบด้วยครับ)

การย้ายข้อความ

ผลของการย้ายข้อความจะเป็นดั

รูปที่ 4 ซึ่งเมื่อพิจารณาแล้วจะเห็นว่าได้มีการปฏิบัติดังนี้

1. ตัดข้อความในบรรทัดปลายทางออกไป ตอกับเศษของตอ

2. ตัดข้อความในบรรทัดต้นบล็อกออกเป็น ท่อนหัวกับเศษของหัว

3. ตัดข้อความในบรรทัดท้ายบล็อกออกเป็น ท่อนหางกับเศษของหาง

4. นำท่อนหัวไปต่อกับตอที่บรรทัดเดิม (2)

5. นำเศษของตอไปต่อกับท่อนหางที่บรรทัดท้ายบล็อก (11)

6. นำเศษของหางไปต่อกับเศษของหัวที่บรรทัดต้นบล็อก (8)

7. ย้ายข้อความในบรรทัดที่ 9, 10 และ 11 ไปแทรกหลังบรรทัดต้นทาง ตามลำดับ

การลอกข้อความ

หากเป็นการลอกข้อความ จะมีการดำเนินการใกล้เคียงกับการย้ายข้อความมาก กล่าวคือ

1. คงเดิม
2. ลอกข้อความเฉพาะท่อนหัว
3. ลอกข้อความเฉพาะท่อนหาง
4. คงเดิม
5. นำเศษของตอไปต่อท้ายท่อนหาง (แล้วเก็บไว้หรือ) นำไปแทรกหลังบรรทัดต้นทาง
6. งด
7. เป็นการลอกข้อความในบรรทัดที่ 9 และ 10 ไปแทรกหลังบรรทัดต้นทางตามลำดับ (ซึ่งจะอยู่หน้าบรรทัดที่มีท่อนหางต่อกับเศษของตอ)

การลบข้อความ

ถ้าเป็นการลบข้อความจะมีขั้นตอนการทำงานน้อยกว่า แต่ผมจะให้หัวข้อตามเดิมเพื่อเปรียบเทียบกันคือ

```
12345678901234567890123456789012345678901234567890123456789012345678901234567890
2aaaaaaaaaaa<aX>XXXXXXXXXXXXXXXXXXXXX8
9.....().....9
10.....(n-nnnnnnnnnn-n).....o....10
11yyyyyyyyyyyyyyyyyyyyyyyyyyy<yA>AAAAAAAAAAAAAAAAAAAAA2
3ooooooooooooooooooooooooooooooooooooooooo3
4ooooooooooooooooooooooooooooooooooooooooooooooooooooooooo4
5bbbbbbbbbbbbbbbb<bB>BBBBBBBBBBBBBBBBBBB5
6ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc6
7ccccccccccccccccccccccccccccccccc7
8xxxxxxxxxxxxxxxxxxxx<xY>YYYYYYYYYYYYYYYYYYYYYYYYYYY11
12345678901234567890123456789012345678901234567890123456789012345678901234567890

14  THIS IS THE LAST LINE OF TEXT
```

รูปที่ 4 แสดงผลของการย้ายข้อมูลตามรูปที่ 3

2 ตัดท่อนหัวทิ้ง

3 ลอกเศษของท่อนหาง

6 นำเศษของท่อนหางไปต่อท้ายเศษของท่อนหัวที่บรรทัดต้นบล็อก

7 ยกเลิกบรรทัดที่ 9, 10 และ 11

จะได้ผลดังรูปที่ 4 ตอนล่าง ขอให้สังเกตด้วยว่าการตัดข้อความในทุกบรรทัดที่กล่าวมา จะเป็นการตัดท่อนท้ายของบรรทัดทั้งสิ้น (เทียบได้กับการตัดปลายเหลือโคนไว้)

แต่การทำงานเป็นบล็อกผู้ใช้อาจหมายกลุ่มข้อความอยู่ในบรรทัดเดียวกันซึ่งจะขอนำไปกล่าวในตัวกิจกรรมโดยตรง หรือเพียง 2 บรรทัด ก็คงมีการดำเนินการเช่นที่กล่าวมาเว้นข้อ 7

ต้น กลาง และท้ายของบรรทัด

เท่าที่กล่าวมานี้เรียกได้ว่าเป็นขั้นตอนวิธี (algorithm) ในการลอก ย้าย และลบข้อความที่เรากำหนดขึ้น (แต่ความจริงนำมาจากเทอร์โบปาสคาล) แต่ยังไม่สมบูรณ์ทีเดียวเพราะได้กำหนดจุดสำคัญทั้งสามจุดไว้กลางบรรทัด ซึ่งในการใช้งานจริงผู้ใช้จะกำหนดไว้ตรงส่วนไหนของบรรทัดก็ได้ และคำทั้งสามนี้มีไม่ครบในบางจุดสำคัญซึ่งจะต้องทราบไว้เป็นพื้นฐานในการเขียนโปรแกรม คือ

ตำบลที่ปลายทาง คือตำแหน่งของจุด dstchn ได้มาจากตำบลที่ของเคอร์เซอร์เมื่อผู้ใช้สั่งให้ลอกหรือย้ายข้อความ (มาตรงจุดนี้ : กดคีย์ Ctrl-K, C หรือ Ctrl-K, V)

ท้ายบล็อก คือตำแหน่งของจุด aftchn ได้มาจากตำบลที่ของเคอร์เซอร์เมื่อผู้ใช้หมายท้ายบล็อก (กดคีย์ Ctrl-K, K)

ทั้งสองจุดนี้มีตำแหน่งที่เป็น ต้น กลาง และท้ายบล็อกครบ นั่นคือ เมื่อแยกบรรทัด (หมายรวมทั้งตัดและลอก) อาจได้ซีกหนึ่งหรือทั้งสองซีกเป็นสตริงว่างกล่าวคือ ได้เป็นสตริงที่มีข้อความทั้งคู่ (กลางบรรทัด) ได้ซีกหนึ่งเป็นสตริงว่าง (ต้นหรือท้ายบรรทัด)

ได้เป็นสตริงว่างทั้งคู่เมื่อจุดดังกล่าวอยู่ต้นบรรทัดว่างเท่านั้น ที่กล่าวเช่นนี้เพราะบรรทัดที่เห็นว่าว่างบนจอภาพและเมื่อเก็บลง buff ก็จะเก็บในฐานะบรรทัดว่าง แต่บนจอภาพจะมีฐานะเป็นบรรทัดของอักขระ space (วรรค) เต็มทั้งบรรทัด (254 อักขระ) ดังนั้นเมื่อกำหนดจุดทั้งสองนี้ตรงใดที่ไม่ใช่ต้นบรรทัด ซีกซ้ายจะเป็นสตริง

```
DESTRUCTOR  Fin;  VIRTUAL;
procedure   FreeStr(sptr : StrPtrT);
function    StrSi(sptr : StrPtrT) : Byte;
procedure   GetMemStr(var tptr : StrPtrT; si : Word);
procedure   CopyStr(var ostr,nstr : StrPtrT; var si : Word);
procedure   ConcatStr(var frst,scnd : StrPtrT; var si : Byte);
procedure   CopyChs(lne : Word; stt,aft : Byte;
                    var nstr : StrPtrT);
procedure   CutChs(lne : Word; stt,aft : Byte; var nstr : StrPtrT);
function    AddChs(lne : Word; istr : StrPtrT) : Boolean;
procedure   ConnectLine(var lne : Word; istr : StrPtrT; var chn : Byte);
procedure   InsChs(lne : Word; stt : Byte; istr : StrPtrT);
procedure   DelChs(lne : Word; stt,aft : Byte);
```

รูปที่ 5

ของอักขระ space ไม่ใช่สตริงว่าง ทั้งนี้ให้หมายรวมถึงการที่ผู้ใช้เลื่อนเคอร์เซอร์ไปท้ายข้อความบนจอภาพด้วย (จะให้เป็นหน้าที่ของวินโดว์อีดิตเป็นผู้ดำเนินการเติม space ให้)

ต้นบล็อก คือตำแหน่งของจุด sttchn ได้มาจากตำบลที่ของเคอร์เซอร์ เมื่อผู้ใช้หมายต้นบล็อก (กดคีย์ Ctrl-K, B)

จุดนี้ไม่มีตำแหน่งปลายบรรทัด คือเมื่อแยกบรรทัดซีกขวาจะต้องเป็นสตริงที่มีข้อความเสมอ โดยอาศัยหลักการเดียวกับที่กล่าวข้างบน เมื่อผู้ใช้กำหนดจุดนี้ตรงใดก็ตาม ซีกซ้ายจะเป็นดังที่กล่าวมา สำหรับซีกขวาจะประกอบด้วยอักขระเท่าที่มีจริงตามด้วยอักขระ space ไปจนสุดบรรทัด (254 อักขระ) นั่นคือไม่มีตำแหน่งปลายบรรทัด

เพื่อไม่ให้สับสนกับการดำเนินการเป็นบรรทัดที่จะกล่าวต่อไป ผมขอทำความเข้าใจไว้ก่อนว่า ต้นบรรทัดและท้ายบรรทัดปกติจะเป็นดังที่เห็นบนจอภาพ เว้นเมื่อมีการดำเนินการดังกล่าวมา

การดำเนินการเป็นบรรทัด

การดำเนินการเป็นบรรทัดสำหรับงานเอดิเตอร์ ได้แก่

- ลบบรรทัด ทั้งบรรทัด (กดคีย์ Ctrl-Y)

- แทรกบรรทัดว่าง (กดคีย์ Ctrl-N ที่ต้นบรรทัด หรือกดคีย์ Enter ที่ท้ายบรรทัดบน)

- ตัดบรรทัด (กดคีย์ Enter หรือ Ctrl-N ที่กลางบรรทัด)

- ต่อบรรทัด (กดคีย์ back space ที่ต้นบรรทัดล่าง)

- ลบบรรทัดจากจุดที่กำหนด (กดคีย์ CtrlQ, Y : งานนี้จะให้วินโดว์อีดิตเป็นผู้ดำเนินการ)

การดำเนินการดังกล่าวมานี้สามารถเขียนโปรแกรมได้โดยตรงเว้นการต่อบรรทัด ซึ่งจะต้องพิจารณาก่อนว่าหากต่อไม่ได้คือ เมื่อต่อแล้วจะได้สตริงยาวกว่า 254 อักขระจะดำเนินการอย่างไร ผมจะให้ไม่ต่อและให้นำไปแทรกไว้ที่บรรทัดหลังบรรทัดของสตริงที่จะนำไปต่อ

การถอนและแทนที่บรรทัด

การถอน (DeleteItem) กับการแทนที่ (ReplaceItem) ตามที่โปรแกรมเขียนไว้ จะมีการดำเนินการเป็นสองขั้นตอนคือ ยกเลิกหน่วยความจำของบรรทัดนั้นในฮีพแล้วถอนหรือแทรกแล้วแต่กรณี

แต่ในบางกรณีเราได้มอบหน่วยความจำหน่วยนี้ให้แก่ตัวแปรสตริง (พอยน์เตอร์) ตัวใหม่แล้วจะให้ยกเลิกไม่ได้ วิธีการคือ ให้ขนาดของบรรทัดนี้เป็นศูนย์ด้วย SetSizeZero แล้วจึงใช้คำสั่ง DeleteItem หรือ ReplaceItem ซึ่งผมจะขอเรียกวิธีการเช่นนี้ว่า "การถอนหรือแทนที่บรรทัดโดยไม่ยกเลิกหน่วยความจำเดิม"

โปรแกรมย่อยภายใน

จากการศึกษาถึงการดำเนินการเป็นบล็อกและเป็นบรรทัดดังกล่าวมา ผมได้นำมาเขียนเป็นโปรแกรมย่อยภายใน คือโปรแกรมที่จะไม่กำหนดเป็นกิจกรรม และไม่ประสงค์ให้ผู้เขียนโปรแกรมคนต่อไปนำไปใช้ และได้บอกใบ้ไว้โดยระบุชื่อไว้หลัง constructor Fin ดังแสดงในรูปที่ 5 และ 7 (เพื่อให้อยู่ใกล้คำอธิบายครับ) คือ

FreeStr

ยกเลิกหน่วยความจำของสตริง sptr ในฮีพ โดยตรวจว่าถ้าไม่ใช่สตริงว่างจึงจะยกเลิก

StrSi

เป็นฟังก์ชันให้ขนาด (ความยาวบวกหนึ่ง) ของสตริง sptr โดยหาความยาวของสตริง ถ้าได้ค่าเป็นศูนย์

จะให้ขนาดเป็นศูนย์ นอกนั้นให้เป็นขนาดบวกหนึ่ง (นั่นคือจะไม่มีค่าขนาดเป็น 1)

GetMemStr

จองหน่วยความจำในฮีพขนาด si ให้แก่ tptr โดยตรวจว่าฮีพว่างหรือไม่ก่อน

*โปรแกรมย่อยทั้งสามนี้เขียนขึ้นเพื่อให้เป็นโปรแกรมย่อยโดยแท้ คือเพื่อให้เขียนโปรแกรมได้สั้น

CopyStr

ลอกข้อความจากสตริง ostr ให้แก่สตริง nstr และให้ค่าขนาดของสตริงนี้แก่ si (เพื่อไม่ต้องหาค่านี้ซ้ำ) โดย

- หาขนาดของสตริง ostr ด้วย StrSi ให้แก่ si
- หากขนาดเป็นศูนย์แสดงว่าเป็นสตริงว่าง ให้ nstr เป็นสตริงว่างด้วย (ให้ชี้ไปที่ nullLn)
- นอกนั้นให้จองหน่วยความจำด้วย GetMemStr ให้แก่ nstr แล้วให้ค่า ostr แก่ nstr

ConcatStr

ต่อสตริง scnd เข้ากับสตริง frst (ให้ frst อยู่ต้นสตริงและให้ค่าแก่ frst) ยกเลิกหน่วยความจำทั้งของ frst เดิม และ scnd ให้ค่าขนาดของสตริงใหม่นี้แก่ si โดยมีเงื่อนไขว่า

- ถ้าสตริงที่จะนำมาต่อคือ scnd เป็นสตริงว่าง (ซึ่งจะต่อหรือไม่ต่อมีค่าเท่ากัน) จะถือว่าไม่ต่อและให้ค่า si เป็นศูนย์
- ถ้าความยาวของสตริงรวมกันเกินกว่า 254 อักขระ จะถือว่าต่อไม่ได้ ให้ค่า si เป็น 255 และจะไม่ดำเนินการใด ได้สตริง frst และ scnd กลับไปตามเดิม

```
PROCEDURE  StrCollect.ConcatStr(var frst,scnd : StrPtrT;
                                var si : Byte);
VAR  tem : StrPtrT;  len1,len2 : Byte;
BEGIN
  si := Length(scnd^);
  if  si = 0  then  Exit  else  len2 := si;

  if  Length(frst^) = 0  then
    begin  frst := scnd;  si := si + 1;  Exit  end
  else  len1 := Length(frst^);

  if  len1+len2 > 254  then  begin  si := 255;  Exit  end;

  si := len1 + len2 + 1;  GetMemStr(tem,si);
  tem^ := frst^ + scnd^;  FreeStr(frst);  FreeStr(scnd);
  frst := tem;
END;
```

รูปที่ 6

- ถ้าต่อได้จะจองเนื้อที่ในฮีพเท่ากับความยาวของสองสตริงรวมกันบวกหนึ่งให้แก่ตัวแปรชั่วคราว ให้ค่า frst+scnd แก่สตริงใหม่นี้ ยกเลิกหน่วยความจำเดิมของ frst และ scnd แล้วโอนสตริงใหม่นี้ให้แก่ frst

*สำหรับการเขียนโปรแกรมคงเป็นไปตามที่กล่าวมานี้ แต่มีเรื่องเบ็ดเตล็ดอยู่บ้าง และเพื่อเป็นตัวอย่างด้วย จึงนำมาแสดงตามรูปที่ 6

*ในโปรแกรมย่อยที่จะกล่าวต่อไปนี้คำว่า stt และ aft หมายถึงอักขระที่

CopyChs

ลอกสตริงในบรรทัดที่ lne ตั้งแต่อักขระ stt จนถึงอักขระก่อน aft ให้แก่ nstr โดย

- ให้ nstr เป็นสตริงว่าง หาขนาดของสตริงเดิมที่บรรทัด lne (ใช้แทนความยาวบวกหนึ่ง)
- หากขนาดเป็นศูนย์ให้กลับ (นั่นคือถ้าสตริงที่จะลอกเป็นสตริงว่าง จะได้ nstr เป็นสตริงว่าง)
- หาก aft มากกว่าขนาดให้ aft เท่ากับขนาด (เพื่อใช้ในการลอกสตริงไปจนสุดสตริง จะได้ไม่ต้องหาความยาวของสตริงเดิม โดยให้ค่าที่ยาวกว่าความยาวของสตริงคือ 255 แก่ aft แทน)
- หาก stt มากกว่าหรือเท่ากับ aft ให้กลับ (โดยปกติจะหมายถึงให้ค่ามาผิด แต่ในบางกรณีจะเป็นการลอกสตริงที่พ้นบรรทัดไปแล้วจึงให้ค่าเป็นสตริงว่าง)
- หาความยาวของสตริงใหม่จากค่า stt ถึงก่อน aft จองหน่วยความจำด้วย GetMemStr ให้แก่ nstr แล้วลอกข้อความจากสตริงเดิมให้แก่ nstr โดยตรง

CutChs

ตัดสตริงในบรรทัดที่ lne ตั้งแต่อักขระที่ stt จนถึงอักขระก่อน aft ให้แก่ nstr โดย

- ลอกข้อความดังกล่าวจากสตริงเดิมด้วย CopyChs ให้แก่ nstr
- หากเป็นสตริงว่าง (ด้วยเหตุผลที่กล่าวใน CopyChs) ให้กลับ
- ลอกท่อนแรกของสตริงเดิมคือตั้งแต่ 1 จนถึงก่อน stt ด้วย CopyChs ให้แก่ตัวแปรตัวหนึ่ง
- ลอกท่อนหลังของสตริงเดิมคือ

ตั้งแต่ aft จนถึงก่อน 255 ด้วย CopyChs ให้แก่ตัวแปรอีกตัวหนึ่ง

- นำสตริงทั้งสองนี้มาต่อกันแล้วนำไปแทนที่สตริงเดิมด้วย ReplaceItem

*ผลของการ CutChs คือสตริงเดิมที่ lne จะถูกตัดคือไม่มีข้อความที่ลอกให้แก่ nstr

*การตัดสตริงนี้จะตัดตรงส่วนไหนของบรรทัดก็ได้ (ต้น กลาง ท้าย)

*จะเห็นว่าในการเขียนโปรแกรมย่อยนี้ได้นำโปรแกรมย่อยที่เขียนไว้มาใช้ซ้อน ๆ กัน การเขียนโปรแกรมแบบนี้เขียนง่าย อ่านเข้าใจง่าย ซอร์สโปรแกรมสั้น แต่เวลาที่ใช้ในการรันจะมากกว่าเขียนขึ้นโดยตรงใหม่

```
procedure   CopyLne(var src,dst : Word);
procedure   MoveLne(var src,dstn : Word);
procedure   NlineCopy;
procedure   NlineMove;
procedure   NlineDel;

procedure   ConnectHTleft;
procedure   CopMveHeadTail(move : Boolean);
procedure   OneLineCopyMove(move : Boolean);
procedure   BlockCopyMove(move : Boolean);
function    DstnInBlock : Boolean;
                END;
```

รูปที่ 7

AddChs

*ทำหน้าที่เป็นผู้เรียกใช้ ConcatStr อีกทอดหนึ่ง โดยให้ผู้ใช้สามารถอ้างถึงสตริงแรกเป็นบรรทัดที่ได้ เรียกว่าเป็นการต่อบรรทัด คือ

เป็นฟังก์ชันนำสตริง istr ไปต่อท้ายสตริงเดิม ณ บรรทัดที่ lne หากความยาวเกินจะไม่ต่อและให้ตนเองเป็นเท็จโดย

- อ่านค่าพอยน์เตอร์ไปยังสตริงเดิมด้วย GetItem ให้แก่ตัวแปรตัวใหม่
- ต่อสตริงด้วย ConcatStr

*เมื่อ ConcatStr ต่อสตริงจะเกิดผล 3 แบบคือ

- ไม่ได้ต่อเนื่องจาก istr เป็นสตริงว่าง จะให้ค่าความยาวของสตริงคืนมาเป็นศูนย์ ซึ่ง AddChs ก็ไม่ต้องดำเนินการใดเช่นกันเพราะสตริงเดิมคงอยู่ตามเดิม
- ต่อได้และต่อมาให้ในตัวแปรตัวใหม่นี้ พร้อมทั้งส่งค่าความยาวรวมมาให้ซึ่ง AddChs ต้องนำมาแทนที่สตริงเดิม ด้วยวิธีแทนที่โดยไม่ยกเลิกหน่วยความจำเดิมด้วย SetSizeZero และ ReplaceItem
- ต่อไม่ได้เนื่องจากความยาวเกินจะส่งสตริงทั้งสองคืนมาและให้ค่าความยาวเป็น 255 ซึ่ง AddChs จะไม่ดำเนินการใดเช่นกัน เพราะสตริงเดิมคงอยู่และสตริง istr ก็ยังคงอยู่จึงให้ตนเองเป็นเท็จเพื่อแจ้งให้ทราบ

ConnectLine

*การต่อสตริงด้วย AddChs มีไม่น้อยที่ต้องการว่าหากต่อไม่ได้ (ความยาวเกิน) ให้แทรกเป็นบรรทัดใหม่หลังบรรทัดที่จะนำไปต่อ จึงได้เตรียมโปรแกรมย่อยนี้ไว้ และจะเรียกว่าเป็นการต่อหรือแทรกบรรทัด คือ

นำสตริง istr ไปต่อท้ายสตริงเดิม ณ บรรทัดที่ lne หากต่อไม่ได้จะเพิ่มหนึ่ง lne และแทรก istr เป็นบรรทัดใหม่ ณ lne ใหม่ ส่งค่า chn คืนมาเป็นตำแหน่งที่ของต้น istr เมื่อต่อแล้วหากแทรกบรรทัดจะได้ chn เป็น 1 โดย

- ให้ขนาดของสตริงที่ lne แก่ chn
- เรียกใช้ AddChs เพื่อการต่อบรรทัด
- หากต่อไม่ได้ให้เพิ่มหนึ่ง lne ให้ chn เป็น 1 และแทรก istr เป็นบรรทัดใหม่ที่ lne

InsChs

นำสตริง istr ไปแทรกสตริงเดิม ณ บรรทัดที่ lne หน้าอักขระที่ stt หากแทรกไม่ได้ (ความยาวเกิน) ให้ตัดข้อความจาก stt ไปจนสุดบรรทัดของสตริงเดิมนำมาต่อท้าย istr และแทรกเป็นบรรทัดใหม่หลังบรรทัดเดิม โดย

- ตัดข้อความในสตริงเดิมจาก stt ให้แก่ตัวแปรด้วย CutChs
- นำมาต่อท้าย istr ด้วย ConcatStr
- นำ istr ใหม่ไปต่อสตริงเดิมที่เหลือหรือแทรกเป็นบรรทัดใหม่ด้วย ConnectLine

DelChs

ลบข้อความในสตริงเดิมที่บรรทัด lne จากอักขระที่ stt ถึงอักขระก่อน aft โดย

- ตัดข้อความดังกล่าวด้วย CutChs ให้ตัวแปรชั่วคราว
- ยกเลิกตัวแปรชั่วคราว

*โปรแกรมย่อยที่กล่าวมาทั้งหมดนี้เรียกได้ว่าเป็นโปรแกรมขั้นพื้นฐานใช้ทั้งในการทำงานเป็นบล็อกและเป็นบรรทัดสำหรับการทำงานเป็นบล็อกจำเป็นต้องจัดโปรแกรมย่อยขึ้นอีกดังแสดงในรูปที่ 7 คือ

CopyLne

ลอกข้อความทั้งบรรทัดจากบรรทัดต้นทางคือ src ไปบรรทัดปลายทางคือ dst โดย

- ให้ค่าพอยน์เตอร์ไปยังสตริงของ src แก่ตัวแปรตัวใหม่ให้ชื่อเป็น sptr ด้วย GetItem
- นำไปแทรกที่ dst ด้วย InsertLine (ซึ่ง InsertLine จะกำหนดตัวแปรคือเนื้อที่ในฮีพ ลอกข้อความจาก sptr คือข้อความที่ src ให้ แล้วจึงนำไปแทรกบรรทัด)

MoveLne

ย้ายข้อความทั้งบรรทัดจากบรรทัดต้นทางคือ src ไปยังบรรทัดปลายทางคือ dst โดย

- ให้ค่าพอยน์เตอร์ไปยังสตริงของ src แก่ตัวแปรตัวใหม่คือ sptr ด้วย GetItem
- นำไปแทรกที่ dst ด้วย InsertItem
- ถอนข้อมูลที่ src โดยไม่ยกเลิกหน่วยความจำเดิมด้วย SetSizeZero และ DeleteItem

*การดำเนินการเช่นนี้เท่ากับเป็นการนำเฉพาะค่าพอยน์เตอร์ไปแทรกเท่านั้น ไม่มีการเคลื่อนย้ายข้อความจริง ๆ ดังเช่นใน CopyLne

NlineCopy

ลอกข้อความทั้งบรรทัดในบล็อก (เศษของบรรทัดต้องดำเนินการก่อน) ตั้งแต่บรรทัดที่ sttlne ถึงบรรทัดที่ aftlne ไปเริ่มลง ณ บรรทัดที่ dstlne โดยทำงานเป็นวงรอบมีจำนวนครั้งเท่ากับจำนวนบรรทัดที่จะลอกในวงรอบมีการทำงานดังนี้

- ลอกข้อความจาก sttlne ไป dstlne ด้วย CopyLne
- เพิ่มหนึ่ง sttlne และ dstlne (ดังนั้นเมื่อจบการทำงานแล้ว dstlne จะอยู่ท้ายบล็อกใหม่)

*ในกรณีที่ dstlne อยู่หน้าบล็อกเมื่อลอกข้อความไปลงคือแทรกด้วย InsertLine จะมีการเพิ่มหนึ่ง sttlne ไปครั้งหนึ่งแล้ว

NlineMove

ดำเนินการเช่นเดียวกับ NlineCopy แต่เป็นการย้าย คือข้อความในบล็อกเดิมจะถูกลบ ทำให้ภายในวงรอบมีการทำงานดังนี้

- ย้ายข้อความจาก sttlne ไป dstlne ด้วย MoveLne
- ถ้า dstlne อยู่หน้า sttlne ให้เพิ่มหนึ่ง dstlne

*ในกรณีที่ dstlne อยู่หลัง sttlne เมื่อยกเลิก sttlne ด้วย DeleteItem จะทำให้บรรทัดที่ dstlne ตัวจริงเลื่อนไปข้างหน้าหนึ่งบรรทัด หากต้องการให้ dstlne ชี้ที่สตริงเดิมจะต้องลดหนึ่งและเพิ่มหนึ่งเพื่อชี้ที่สตริงถัดไป จึงไม่ต้องดำเนินการใด สำหรับ sttlne เมื่อถอนข้อมูลแล้ว บรรทัดหลังจะเลื่อนมาแทนที่จึงไม่ต้องดำเนินการใดเช่นกัน

NLineDel

ลบข้อความทั้งบรรทัดในบล็อกตั้งแต่บรรทัดที่ sttlne ถึง aftlne โดยทำงานเป็นวงรอบเท่าจำนวนบรรทัดที่จะลบภายในวงรอบก็เพียงแต่ให้ลบข้อความครั้งละ 1 บรรทัดด้วย DeleteItem

OneLineCopyMove

ดำเนินการลอกหรือย้ายข้อความสุดแต่จะให้ move เป็นเท็จหรือจริงตามลำดับ เมื่อผู้ใช้หมายบล็อกไว้ในบรรทัดเดียวกัน โดย

- ตรวจว่าถ้า sttchn มากกว่าหรือเท่ากับ aftchn จะถือว่ามีการผิดพลาดไม่ดำเนินการใด
- หาผลต่างของ aftchn กับ sttchn เก็บไว้
- ถ้าเป็นการย้ายข้อความให้ตัดข้อความเดิมจาก sttchn ถึงก่อน aftchn ด้วย CutChs แต่ถ้าเป็นการลอกให้ลอกด้วย CopyChs
- นำข้อความที่ตัดหรือลอกมานี้ไปแทรกในบรรทัดที่ dstlne เริ่ม ณ ตำแหน่งที่ dstchn ด้วย InsChs
- ปรับแต่งบล็อกใหม่ให้ sttlne และ aftlne เท่ากับ dstlne และให้ sttchn เท่ากับ dstchn ส่วน aftchn จะเท่ากับ sttchn + ผลต่างที่เก็บไว้

CopMveHeadTail

*เมื่อผู้ใช้หมายบล็อกหลายบรรทัดคือมีท่อนหัวอยู่ที่บรรทัดหนึ่งและท่อนหางอยู่อีกบรรทัดหนึ่ง จะต้องมีการดำเนินการดังที่กล่าวไว้ในหัวข้อการดำเนินการเป็นบล็อก แต่การลอกและย้ายข้อมูลเป็นบล็อกมีขั้นตอนของการทำงานใกล้เคียงกันมาก ผมจึงนำมารวมกันเป็น

1. ตัดบรรทัดปลายทางเป็นตอกับเศษของตอ
2. ลอกข้อความที่เป็นท่อนหัว
3. ลอกข้อความที่เป็นท่อนหาง
4. ถ้าเป็นการย้ายข้อความให้ต่อเศษของท่อนหางเข้ากับเศษของท่อนหัว (เพื่อจะได้ไม่ต้องคอยติดตามว่าจะถูกเคลื่อนย้ายไปที่ไหน)
5. ต่อท่อนหัวเข้ากับตอ
6. นำเศษของตอมาต่อเข้ากับท่อนหางแล้วนำไปแทรกไว้หลังบรรทัด

ปลายทาง

7. หากมีการย้ายข้อความทั้งบรรทัดให้นำมาแทรกหลังบรรทัดปลายทางไปตามลำดับ ซึ่งจะอยู่หน้าบรรทัดที่ประกอบด้วยหางและเศษของตอ

*ผมได้แยกข้อ 1 ถึงข้อ 6 มาเขียนเป็น CopMveHeadTail แต่ในการเขียนโปรแกรมจริงมีเทคนิคเพิ่มอีก จึงนำโปรแกรมมาแสดงดังรูปที่ 8 บรรทัดที่ 1 ถึง 5 จะเป็นไปดังที่กล่าวมา แต่ในการนำเศษของตอมาต่อเข้ากับท่อนหางนั้น ได้แทรกท่อนหางเป็นบรรทัดใหม่หลังบรรทัดปลายทางก่อนแล้วจึงนำเศษของตอมาต่อ เผื่อล้นบรรทัดจะได้แทรกเป็นบรรทัดใหม่ได้โดยตรง

```
PROCEDURE   StrCollect.CopMveHeadTail(move : Boolean);
VAR   dstlft,head,tail : StrPtrT;   nxtlne : Word; si : Byte;
BEGIN
  CutChs(dstlne,dstchn,255,dstlft);
  CopyChs(sttlne,sttchn,255,head);

  CopyChs(aftlne,1,aftchn,tail);
  if  move  then  ConnectHTleft;

  ConnectLine(dstlne,head,sttchn);

  nxtlne := dstlne + 1;
  ok := InsertItem(nxtlne,StrSi(tail),tail);
  ConnectLine(nxtlne,dstlft,aftchn);
END;
```

รูปที่ 8

ConnectHTleft

ต่อเศษของท่อนหางเข้ากับเศษของท่อนหัวที่บรรทัดเดิมของท่อนหัวโดย

- ตัดข้อความที่เป็นท่อนหัวทิ้ง
- ถ้า aftchn = 1 คือท้ายบล็อกอยู่ท้ายบรรทัดบนจึงไม่มีเศษของท่อนหางไม่ต้องดำเนินการต่อไป
- ลอกเศษของท่อนหาง นำมาต่อกับเศษของท่อนหัว
- ยกเลิกบรรทัดที่เป็นท่อนหางทั้งบรรทัด
- ถ้า aftlne น้อยกว่า dstlne ให้ลดหนึ่ง dstlne เพื่อชดเชย

*โปรแกรมนี้เขียนขึ้นเพื่อใช้กับการลบข้อความโดยตรง แต่ให้การย้ายข้อความนำไปใช้ได้ด้วยโดยให้ลอกข้อความที่เป็นท่อนหัวไว้ก่อน

BlockCopyMove

ดำเนินการลอกหรือย้ายข้อความเป็นบล็อกตามแต่จะให้ move เป็นเท็จหรือจริง โดย

- หาผลต่างของ aftlne กับ sttlne ให้แก่ dif
- ถ้า dif = 0 (บล็อกอยู่ในบรรทัดเดียวกัน) ให้ดำเนินการโดยเรียกใช้ OneLineCopyMove แล้วให้กลับ
- ดำเนินการกับท่อนหัวและหางด้วย CopMveHeadTail
- เพิ่มหนึ่ง dstlne กับ sttlne และลดหนึ่ง aftlne
- ถ้า dif>1 ให้ทำ NlineCopy หรือ NlineMove แล้วแต่กรณี
- ปรับแต่งบล็อกใหม่โดยให้ aftlne = dstlne (ตามที่กล่าวใน NlineCopy) ให้ sttlne = aftlne-dif แล้วย้อนให้ dstlne = sttlne และให้ dstchn = sttchn (ส่วนค่าของ sttchn กับ aftchn นั้น ConnectLine เป็นผู้ดำเนินการให้ใน CopMveHeadTail)

DstnInBlock

เป็นฟังก์ชันให้ค่าความเป็นจริงเมื่อตำบลที่ปลายทาง (dstlne, dstchn) อยู่ในบล็อก

กิจกรรมหลัก

จากโปรแกรมย่อยดังกล่าวมานี้ได้นำมาเขียนโปรแกรมของกิจกรรมหลักตามที่กำหนดไว้ในรูปที่ 9 (ในโปรแกรมจริง การระบุกิจกรรมส่วนนี้จะอยู่ระหว่างรูปที่ 2 กับ 5) ได้แก่

BlockCopy

BlockMove

ลอก/ย้ายข้อความในบล็อกไปยังตำบลที่ปลายทางซึ่งหมายไว้ด้วย dstlne กับ dstchn โดย

- ถ้าบล็อกไม่ถูกต้องจะให้กลับโดยไม่ดำเนินการใด
- เรียกใช้ BlockCopyMove โดยให้ค่าเท็จ/จริงแก่ move

BlockDelete

ถอนข้อความในบล็อก โดย

- ถ้าบล็อกไม่ถูกต้องให้กลับโดยไม่ดำเนินการใด
- หาผลต่างของ aftlne กับ dstlne ให้แก่ dif
- ถ้าตำบลที่ปลายทางอยู่ในบล็อกให้เลื่อนไปไว้ต้นบล็อก
- ถ้าตำบลที่ปลายทางอยู่หลังบล็อก ให้ลดค่าใน dstlne ด้วยค่า dif-1 (เพื่อปรับแต่ง dstlne ให้ถูกต้อง)
- ถ้า dif = 0 แสดงว่าบล็อกอยู่

ภายในบรรทัดเดียวกันให้ลบข้อความด้วย DelChs

- นอกนั้นให้ดำเนินการกับท่อนหัวและท่อนหางที่เหลือด้วย Connect-HTleft

- และถ้า dif มากกว่า 1 ให้เพิ่มหนึ่ง sttlne และลดหนึ่ง aftlne และลบบล็อกที่เหลือทีละบรรทัดด้วย NlineDel

- ให้ sttlne := aftlne+1 (เพื่อแสดงว่าไม่มีการหมายบล็อก)

GetHead

GetTail

เป็นฟังก์ชันที่เตรียมไว้ให้ผู้เขียนโปรแกรมคนต่อไปอ่านข้อความที่เป็นท่อนหัว/ท่อนหาง (เช่นในงานพิมพ์หรือเก็บลงไฟล์อื่น) และให้แจ้งโดยให้ตนเองเป็นเท็จเมื่อข้อความที่อ่านได้เป็นสตริงว่าง

DoEnter

เพื่อการขึ้นบรรทัดใหม่เช่นเมื่อผู้ใช้กดคีย์ Enter จะดำเนินการโดย

- ตัดข้อความ ณ บรรทัดที่ dstlne จากอักขระที่ dstchn ไปจนสุดบรรทัดด้วย CutChs ให้แก่ sptr (ซึ่งถ้า dstchn อยู่ท้ายบรรทัดก็จะได้ sptr เป็นสตริงว่าง)

- เพิ่มหนึ่ง dstlne และให้ dstchn เป็น 1

- แทรกสตริง sptr ที่บรรทัด dstlne ด้วย InsertItem

DoReEnter

นำข้อความในบรรทัดล่างไปต่อกับบรรทัดบน (เช่นเมื่อผู้ใช้กดคีย์ back space ที่ต้นบรรทัด) โดย

- ให้ตัวแปร sptr ชี้ไปยังข้อความ ณ บรรทัดที่ dstlne ด้วย GetItem

- ยกเลิกบรรทัดที่ dstlne แต่ไม่ยกเลิกหน่วยความจำในฮีพโดย Set-SizeZero ก่อนแล้วจึง DeleteItem

- ลดหนึ่ง dstlne แล้วนำสตริง sptr ไปต่อด้วย ConnectLine (ซึ่งถ้าความยาวเกินก็จะมีสภาพเหมือนเดิม)

```
procedure  BlockCopy;
procedure  BlockMove;
procedure  BlockDelete;
function   GetHead(var nstr : StrPtrT) : Boolean;
function   GetTail(var nstr : StrPtrT) : Boolean;

procedure  DoEnter;
procedure  DoReEnter;
procedure  CheckMem(memreq : Word);
```

รูปที่ 9

ข้อความตัวอย่าง

เรื่องราวของสกุล StrCollect มีเพียงเท่านี้ครับ จะเห็นว่าเขียนโปรแกรมย่อยมากมาย แต่ที่จะนำไปใช้ต่อไปคือในรูปที่ 2 และ 9 เท่านั้น

ผมได้เตรียมข้อความตัวอย่างเพื่อทดสอบกิจกรรมของสกุลนี้ไว้ในไฟล์ DATA. TXT ดังแสดงในรูปที่ 3 โปรดสังเกตว่าผมได้เตรียมบล็อกหลายบรรทัดไว้ 2 บล็อกคือ บล็อก Ab กับบล็อก Xy (ที่ใช้เป็นตัวอย่างในหัวข้อการทำงานเป็นบล็อก) เพื่อการย้ายข้อมูลไปข้างหลังและข้างหน้าตามลำดับ กับบล็อกบรรทัดเดียวอีก 1 บล็อกคือ บล็อก nn ในบรรทัดที่ 10

สำหรับตำบลที่ปลายทางควรเป็น A (ใหญ่) หรือ B (ใหญ่) ตัวแรกเพื่อการย้ายข้อมูลไปข้างหน้า และ X (ใหญ่) หรือ Y (ใหญ่) ตัวแรกเพื่อการย้ายข้อมูลไปข้างหลัง และตรง) ในบรรทัดที่ 9 เพื่อการทำงานในบรรทัดซึ่งถ้าโปรแกรมไม่ผิดพลาดจะได้เครื่องหมายวงเล็บประกบกันถูกต้องดังในรูปที่ 4

ข้อความตัวอย่างนี้ผมดำเนินการด้วยเทอร์โบปาสคาล (และผลของการทำงานในรูปที่ 4 ก็เป็นของเทอร์โบปาสคาล) หากคุณป้อนข้อความเองปลายบรรทัดสุดท้ายจะต้องไม่กดคีย์ Enter ไม่เช่นนั้นจะได้บรรทัดว่างอีก 1 บรรทัด

โปรแกรมทดสอบ

ได้เตรียมโปรแกรมทดสอบกิจกรรมในสกุลไว้ในไฟล์ TSTSTRCL.PAS ซึ่งมีโปรแกรมย่อยสำหรับอ่านข้อความตัวอย่างให้แก่ buff กับโปรแกรมย่อยสำหรับนำข้อความใน buff ออกแสดง ในโปรแกรมหลักได้เตรียมคำสั่งสำหรับกำหนดตำบลที่ปลายทางและหมายบล็อกไว้หลาย ๆ แบบอยู่ในวงเล็บ ( ) เพื่อให้คุณเลือกมาทดสอบด้วยคำสั่งสำหรับการทำงานเป็นบล็อกตามต้องการ

ขอให้ทดสอบกับคำสั่งอื่น ๆ ด้วยครับ โดยเฉพาะคำสั่งที่เดียวกับการทำงานเป็นบรรทัดผมยังไม่ได้ทดสอบเลย

ข้อดีและข้อเสียของโอโอพี

ตามที่ผมกล่าวในสกุล Collection

ว่าจะแสดงข้อดีและข้อเสียในการเขียนโปรแกรมในแบบโอโอพีให้ทราบ ข้อเสียก็คือหากผมต้องการเขียนโปรแกรมสำหรับสกุล StrCollect โดยตรง ผมไม่จำเป็นต้องกำหนดสกุล Collection ขึ้นก่อน สามารถนำกิจกรรมของสกุล Collection มาไว้ในสกุล StrCollect ได้โดยตรง ซึ่งจะประหยัดทั้งหน่วยความจำ (เช่น ไม่ต้องมีค่า size) และเวลาที่ใช้ในการรัน (ไม่ต้องเรียกใช้โปรแกรมย่อยซ้อน ๆ กันและข้ามสกุล)

ส่วนข้อดีดังเช่นสกุล Collection อีกเช่นกัน ผมเขียนทิ้งไว้นานจนกล่าวได้ว่าลืม เมื่อผมจะเขียนสกุล StrCollect ก็เพียงแต่นำบันทึกหน้าที่และการทำงานของกิจกรรมในสกุล Collection มาทบทวน ก็สามารถทำงานต่อได้ทันทีโดยไม่ได้สนใจซอร์สโค้ดเดิมเลย

บางท่านอาจแย้งผมว่าทำเป็นยูนิตไว้ได้เช่นกัน ถูกต้องครับเพราะยูนิตในเทอร์โบปาสคาล มีความเป็นโอโอแอลเหมือนกันแต่เป็นอยู่ข้อเดียวคือความเป็นเอกเทศ

อีกประเด็นหนึ่งคือในการกำหนดกิจกรรมผมจะมุ่งที่ออปเจ็คเป็นหลัก เมื่อออปเจ็คคือข้อความเป็นบล็อกและเป็นบรรทัด ผมจึงมุ่งอยู่ที่จะให้มันทำอะไรได้บ้างโดยไม่คำนึงถึงเรื่องอื่นเลย และในการเขียนโปรแกรมย่อยก็เช่นกัน จะเขียนเพื่อสนองตอบการทำงานของกิจกรรมเท่านั้น ทำให้จำกัดขอบเขตของการเขียนโปรแกรมให้อยู่ในแวดวงอันหนึ่ง

ซอฟต์เมล์

โปรแกรมในซอฟต์เมล์อาจไม่ตรงกับที่ผมกล่าวมาทีเดียวนักหากมีการปรับปรุง และสำหรับผู้ที่สั่งซอฟต์เมล์ของไมโครฯ ฉบับที่ 64 คือที่มีเรื่องของสกุล Collection แล้วไม่ต้องสั่งซ้ำครับ เพราะผมได้ให้โปรแกรมในตอนนี้ไปด้วยคือจะประกอบด้วยไฟล์

COLLECT.PAS
TSTCLCT.PAS
STRCLCT.PAS
TSTSTRCL.PAS
DATA.TXT
PRIME.PAS
EDITROOT.PAS

สองไฟล์หลังสำหรับบทความในตอนหน้า (แต่ยังไม่ครบทุกไฟล์) เพื่อเก็บไว้อ่านประกอบกับบทความครับ

แก้ที่ผิดไมโครฯ 63

เนื่องจากในวารสารไมโครคอมพิวเตอร์ ฉบับที่ 63 ได้พิมพ์ข้อความในบทความเรื่อง **แนะนำเทอร์โบ C++** หน้า 323 ผิดพลาดบางส่วน จึงขอแก้ไขข้อความใหม่ดังนี้

ข้อความเดิม

และในกรณีของไฟล์.h เทอร์โบ-ปาสคาล 5.5 ไม่ได้ให้คอมพายล์สกุล

แก้ใหม่เป็น

และในกรณีของไฟล์.h เทอร์โบ-ปาสคาล 5.5 ให้คอมพายล์สกุล

ข้อความเดิม

บอร์แลนด์ได้ปรับปรุงเทอร์โบ-ปาสคาลของงานให้เขียนโปรแกรมในแบบโอโอพีได้เช่นกันในรุ่น 5.5 และออกมาก่อนหน้าเทอร์โบ C++ ซึ่งในเทอร์โบปาสคาล 5.5 เองก็กล่าวไว้ในหนังสือ "The C++ Programming

เพิ่มเติมข้อความเป็น

บอร์แลนด์ได้ปรับปรุงเทอร์โบ-ปาสคาลของคนให้เขียนโปรแกรมในแบบโอโอพีได้เช่นกันในรุ่น 5.5 และออกมาก่อนหน้าเทอร์โบ C++ ซึ่งในเทอร์โบปาสคาล 5.5 เองก็กล่าวไว้ในคู่มือว่า เป็นไปตามหนังสือ "The C++ Programming.